Expert Systems With Applications 141 (2020) 112963

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Application of deep reinforcement learning to intrusion detection for supervised problems

Manuel Lopez-Martin*, Belen Carro, Antonio Sanchez-Esguevillas

Dpto. TSyCeIT, ETSIT, Universidad de Valladolid, Paseo de Belén 15, Valladolid 47011, Spain

ARTICLE INFO

Article history:
Received 9 May 2019
Revised 26 July 2019
Accepted 17 September 2019
Available online 18 September 2019

Keywords:
Intrusion detection
Data networks
Deep reinforcement learning

ABSTRACT

The application of new techniques to increase the performance of intrusion detection systems is crucial in modern data networks with a growing threat of cyber-attacks. These attacks impose a greater risk on network services that are increasingly important from a social end economical point of view. In this work we present a novel application of several deep reinforcement learning (DRL) algorithms to intrusion detection using a labeled dataset. We present how to perform supervised learning based on a DRL framework.

The implementation of a reward function aligned with the detection of intrusions is extremely difficult for Intrusion Detection Systems (IDS) since there is no automatic way to identify intrusions. Usually the identification is performed manually and stored in datasets of network features associated with intrusion events. These datasets are used to train supervised machine learning algorithms for classifying intrusion events. In this paper we apply DRL using two of these datasets: NSL-KDD and AWID datasets. As a novel approach, we have made a conceptual modification of the classic DRL paradigm (based on interaction with a live environment), replacing the environment with a sampling function of recorded training intrusions. This new pseudo-environment, in addition to sampling the training dataset, generates rewards based on detection errors found during training.

We present the results of applying our technique to four of the most relevant DRL models: Deep Q-Network (DQN), Double Deep Q-Network (DDQN), Policy Gradient (PG) and Actor-Critic (AC). The best results are obtained for the DDQN algorithm.

We show that DRL, with our model and some parameter adjustments, can improve the results of intrusion detection in comparison with current machine learning techniques. Besides, the classifier obtained with DRL is faster than alternative models. A comprehensive comparison of the results obtained with other machine learning models is provided for the AWID and NSL-KDD datasets, together with the lessons learned from the application of several design alternatives to the four DRL models.

1. Introduction

Proper and reliable operation of data networks is crucial in terms of economic and social impacts due to the importance of the telecommunication services deployed in the networks. Given this scenario and the impact of cybersecurity issues, it is important to provide novel algorithms that help detect intrusions and are also efficient in terms of the necessary resources, mainly considering the new nature of networks which are massive and complex, such as IoT networks. It is possible to classify these algorithms by detection approach in signature-based detection and anomaly-based detection (Bhuyan, Bhattacharyya, & Kalita, 2014). Signature-based detection uses a database of previously identified bad patterns to report an attack; while anomaly-based detection uses a machine learning model to classify traffic as good or bad, based on a training dataset of features (usually based on traffic flows) and their corresponding labels (attack categories).

For this work, we have applied anomaly-based supervised machine learning (ML) models to two different, and, well-known, intrusion detection datasets: the NSL-KDD dataset (Tavallaee et al., 2009) and the AWID dataset (Kolias et al., 2016).

A supervised ML model applies a generic learning algorithm that tunes a number of parameters to optimize prediction. At the center of this learning process is a dataset of recorded samples. These samples are formed by a vector of features (network features in our case) with associated pre-assigned labels. These labels are the elements that the algorithm must predict for a new vector of features.

* Corresponding author.
E-mail addresses: mlopezm@ieee.org, mlopezm@acm.org (M. Lopez-Martin), belcar@tel.uva.es (B. Carro), antoniojavier.sanchez@uva.es (A. Sanchez-Esguevillas).

https://doi.org/10.1016/j.eswa.2019.112963
0957-4174/

Contrary to the supervised learning framework, in a classic DRL framework (Arulkumaran et al., 2017) there are two basic components: an agent and the environment. The agent can observe the state of the environment and produce an action. The action is received by the environment generating a new state and a reward, which is associated to the good or bad result of the action. The objective of the learning framework is that the agent can learn to deliver an optimized sequence of actions that maximizes the total sum of rewards. In a classic DRL framework there is no stored dataset of features (states) with their associated actions (label). Instead, the framework attempts to learn this correspondence through an intermediate function (the reward function), and learning is not based on instantaneous information about best actions (feature samples and labels in a training dataset) but on a series of indications (rewards) on the value of intermediate actions with the ultimate goal of maximizing the total sum of rewards.

One important objective for this work is to show that, with some adaptations, we can apply a DRL algorithm using a dataset of labeled samples without interacting with a live environment, as required by the classic DRL framework, and to present in detail the adjustments needed to perform these adaptations. With this aim, we begin by assimilating the actions with the intrusion labels and the states with the network features. The environment is simulated in such a way that it responds with a positive reward in case of positive detection and a negative one in the opposite case. Any new state provided by the environment corresponds to the sampled features of the training dataset. An important result is that the succession of states does not form a sequence since the samples in the dataset are usually independently distributed. This is not the case for classic DRL problems, where the states form a natural sequence and the next state depends on previous states and actions. Nevertheless, the machinery of DRL algorithms can be applied also in this case, but we must take special care with the values of some parameters if we want to obtain good results. Such a parameter is the discount factor (Sutton & Barto, 1998), whose value must be very low, contrary to the value that is generally found in classic DRL configurations. Section 4 shows the great influence of the value of the discount factor on the detection metrics and Section 3.2.2 gives the reason for that impact.

When applying DRL we can use different algorithms based on different problem requirements and assumptions. We have analyzed the adequacy of four DRL algorithms to our problem: Deep Q-Network (DQN) (Mnih et al., 2013), Double Deep Q-Network (DDQN) (Van Hasselt et al., 2015), Policy Gradient (PG) (Bartlett & Baxter, 2011) and Actor-Critic (AC) (Grondman et al., 2012). In Section 4 a comprehensive comparison of the results obtained from these models, together with results obtained from other commonly applied machine learning models is provided, considering different common metrics: precision, F1, accuracy and recall. The conclusion of this comparison is that our proposed framework, using the DDQN algorithm, offers a prediction performance better or similar to the state-of-the-art (SOTA) models, for the two datasets, with the additional advantage of being faster at prediction time.

The DRL models for intrusion detection presented here show many advantages over other ML models. The main contributions of this work are to demonstrate these advantages and present them as a good alternative to other ML models. The list of advantages are: (1) the neural networks used to implement the classifier: Policy, Value or Q functions, are simple and fast, which makes them appropriate for new networks with demanding requirements, such as Internet of Things (IoT) Networks (Zarpelo et al., 2017); (2) the resulting neural network is suitable for distributed high-performance computing environments (e.g. Tensorflow); (3) the reward function used to drive detection can be extremely flexible and does not need to be differentiable; (4) the nature of the model allows a simple update of the parameters learned in case of new data available (on-line learning).

The paper is organized as follows: Section 2 presents related works, Section 3 describes the work performed, Section 4 shows and compares the results, and finally Section 5 offers discussion and conclusions.

2. Related works

The NSL-KDD and AWID datasets have been widely used in the literature, applying many algorithms. This section presents the most illustrative works of intrusion detection applied to these datasets, along with a general discussion about works on ML for cybersecurity with the recent use of reinforcement learning for IDS in particular.

There are a number of excellent reviews on the taxonomy, status and current developments of ML for cybersecurity (Bhuyan et al., 2014; Hussain et al., 2019), which is a sub-area of the more generic application of ML to the analysis and prediction of network traffic (Boutaba et al., 2018; Wang et al., 2018).

ML supervised (da Costa et al., 2019) and unsupervised (Nisioti et al., 2018) methods have been widely applied to intrusion detection for data networking, which is currently and area of active research. The classic ML models applied to IDS have been (Tsai et al., 2009; Yavanoglu & Aydos, 2017): Support Vector Machine (SVM), Multilayer Perceptron (MLP), K-Nearest Neighbors (KNN), Decision Trees (DT), Naive Bayes (NB) and Random Forest. It is important to mention the increasing use of the new deep learning models, in particular convolutional and recurrent networks and generative models (Berman et al., 2019; Chalapathy & Chawla, 2019; Kim & Aminanto, 2017)

The NSL-KDD dataset is a classic intrusion detection dataset used in a multitude of previous and recent research works. Lopez-Martin et al. (2019) presents an extensive list of references to works related with intrusion detection for the NSL-KDD dataset. It is difficult to compare results, since many works use different test sets to those proposed by NSL-KDD, nevertheless it can be noticed that an accuracy close to 80% (5 labels) and 90% (2 labels) can be considered as state-of-the-art results. Some additional and more recent results to these are given in Chen et al. (2018) that presents a convolutional autoencoder for the NSL-KDD with 2-labels attaining an accuracy of 96.87%. In the same line, Shone et al. (2018) presents a novel stacked non-symmetric deep autoencoder with an overall accuracy (5-labels) of 97.85%. Woo, Song, and Choi (2019) applies feature selection and a correct MLP layer configuration to achieve an average accuracy of 98.5% for the NSL-KDD. The excellent review of Berman et al. (2019) presents the application of different configurations of autoencoders, MLPs, recurrent networks and restricted Boltzmann machines to NSL-KDD with different results in prediction accuracy. Thomas and Pavithran (2018) provides a review of 8 works using NSL-KDD with different ML models and data preparation processes, with a maximum accuracy of 99.81% for a hybrid model of J48, Random Tree, REPTree, AdaBoostM1, Decision Stump and NB. In Javaid et al. (2016) self-taught learning (STL) is proposed as a model based on deep learning that achieves a F1 score, for 2 labels, of 90.4%; STL consists of an initial dimensionality reduction model that is trained with labeled and unlabeled data to obtain a good feature representation; this initial model is used later to train a common classifier, using the new features produced by the initial model as input to the classifier. The work in Bhattacharjee, Fujail, and Begum (2017) uses two types of clustering algorithms (K-Means and Fuzzy C-Means) for intrusion detection in NSL-KDD (5-labels) with the best attack detection of 45.95% for Fuzzy C-Means.

The work that originally presented the AWID dataset (Kolias et al., 2016) provides a comprehensive analysis of the

performance of different machine learning algorithms for predicting attacks. This work applies 8 classification algorithms to AWID, being the J48 model (an implementation of C4.5) that stands out in the results with 96% (accuracy) and 95% (F1-score). Considering more recent results, Wang et al. (2019) employs Stacked Autoencoder (SAE) and Deep Neural Networks (DNN) to perform classification of the four types of AWID attacks with accuracies from 99.9% to 73.1%. The review in Berman et al. (2019) presents also and autoencoder applied to AWID. Abdulhammed et al. (2018) applies feature selection and seven classic ML models to AWID, with Random Forest providing the best accuracy results (99.64%). Rezvy et al. (2019) achieves an overall accuracy of 99.9% for AWID with a deep autoencoder. The work of Qin et al. (2018) proposes a fast algorithm based on SVM applied to the AWID dataset with a reduced number of features, obtaining an accuracy between 87.34% and 99.98% for the different attack types. Nivaashini and Thangaraj (2019) review common ML models applied to intrusion detection for wireless networks; in this study, AWID is introduced as the reference dataset for these networks. In Moshkov (2017) gradient boosting, Random Forest and MLP models are applied to intrusion detection for wireless networks, proposing AWID as the chosen dataset with the best results obtained for gradient boosting. The work in Thanthrige, Samarabandu, and Wang (2016) explores the importance of feature selection using information gain and Chi-squared statistics, achieving an improvement of 2.4% after feature reduction using Random Tree with a final overall accuracy of 95.12%.

There are works performing intrusion detection in a simulated or real environment, applying a reinforcement learning algorithm with Q-learning, and relying on look-up tables and temporal differences, but not DRL (Sukhanov et al., 2015). Similarly, Xu (2006) applies a kernel version of temporal differences to a Markov chain prediction problem and Xu and Luo (2007) applies the same model to a host-based IDS. An RL multi-agent scenario for intrusion detection has been used in Servin (2009), also making use of look-up tables to implement the algorithms in a simulated network.

Deokar and Hazarnis (2012) proposes a framework for using log-files at different system levels (e.g. client, proxy, firewall, network and system) based on a hierarchy of log correlations implemented with association rules (AR), where the strength of each AR is controlled by a reinforcement learning model. The framework assumes interaction with a live environment.

It is worth mentioning the interest of reinforcement learning models applied to cyber-physical-systems (CPS) (i.e. evolution of embedded systems with highly integrated physical, computer-based and network elements). In this field, Feng and Xu (2017) presents an application of DRL (actor critic) to a CPS system with a dynamical system perspective where the objective is to defend the system against cyber-attacks. In the same area, Akazaki et al. (2018) introduces a study to prevent the falsification of entries to a CPS using DRL with the intention of generating 'counterexamples' that could endanger the system. The detection of these 'counterexamples' is critical to protect a system against possible attacks, in the same line presented in the study of Goodfellow, Shlens, and Szegedy (2015) to guarantee the robustness in the classification of images.

There are excellent reviews of DRL in general (Francois-Lavet et al., 2018; Li, 2018) that present the different methods and current applications in diverse fields. Li (2018) shows current DRL applications for security, with special emphasis on adversarial reinforcement learning. Caminero et al. (2019) also presents a recent multi-agent adversarial reinforcement learning model for IDS.

Nguyen and Reddi (2019) offers a complete and updated review of DRL for Cybersecurity, the works presented are based on a live environment (real or simulated).

3. Work description

This Section presents the different elements that integrate the work: the datasets used for the experiments and the different DRL models employed. The datasets are described in Section 3.1. The proposed algorithms, with our framework adaptations, are presented in detail in Section 3.2.

3.1. Intrusion detection datasets

For this work, we have considered two datasets that satisfy a series of requirements: (1) labeled datasets, (2) unbalanced but with a different level of imbalance, which allows studying the behavior in different conditions, (3) a predefined split for the training and test sets, which offers a means to compare results from different works, (4) well-known datasets, which make available a sufficient number of results from previous works, (5) to include older and more recent datasets, to increase generality/variability, (6) data coming from different network architectures (e.g. fixed-line vs wireless networks), and (7) the need to have a data volume large enough to have significant results, but limited by practical restrictions of memory and CPU time. Considering these aspects, we have opted for the NSL-KDD and AWID datasets, which satisfy most of the listed requirements (Ring et al., 2019).

In addition, NSL-KDD and AWID are among the most frequently used intrusion detection datasets, in two recent literature reviews: Ring et al. (2019) and da Costa et al. (2019). This importance can also be inferred from the significant number of current works related to both datasets; as a summary: Berman et al. (2019), Chen et al. (2018), Javaid et al. (2016), Nivaashini and Thangaraj (2019), Shone et al. (2018), Thanthrige, Samarabandu, and Wang (2016), Moshkov (2017), Rezvy et al. (2019), Qin et al. (2018), Abdulhammed et al. (2018), Thomas and Pavithran (2018), Bhattacharjee, Fujail, and Begum (2017), Yavanoglu and Aydos (2017), and Woo et al. (2019).

3.1.1. NSL-KDD

The NSL-KDD is a paradigmatic and frequently used IDS dataset (Tavallaee et al., 2009). It contains a moderately large number of samples (>120K samples) with continuous and categorical features (>40 features).

A good description of the NSL-KDD dataset is provided in Lopez-Martin et al. (2019). One interesting aspect of NSL-KDD is that the test and training sets have a different distribution of labels, which is an important challenge for any classifier. The dataset preparation and scaling for this work is similar to that of Lopez-Martin et al. (2019) with the difference that in this case we have aggregated the labels in two categories: normal and anomaly. This final grouping makes sense, since with the NSL-KDD dataset we want to verify the capacity of the classifier to handle a different distribution of attacks between the training and test dataset. While the AWID dataset has been used to verify the ability to handle a very unbalanced dataset.

In Fig. 1 is presented the frequency of these aggregated categories in the NSL-KDD training and test datasets.

3.1.2. AWID

Aegean Wi-Fi Intrusion Dataset (AWID) (Kolias et al., 2016) is a public dataset containing normal traffic along with three types of attacks on IEEE 802.11 networks. From the diverse groups of data provided by AWID, we have employed the AWID-CLS-R dataset with differentiated training and test sets. It has 4 labels to classify: normal, flooding, injection and impersonation; with over 150 categorical and continuous features with a relatively large number of samples for the training (>1,5 M) and test sets (>500 K). The number of features can be reduced to a small set of important ones, as

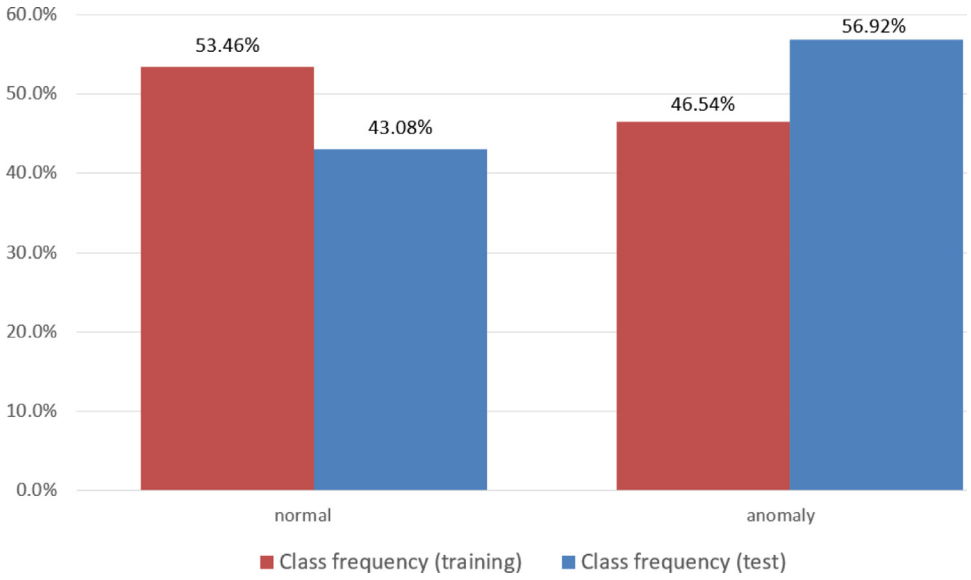


Fig. 1. Frequencies for intrusion categories for the training and test sets (NSL-KDD).

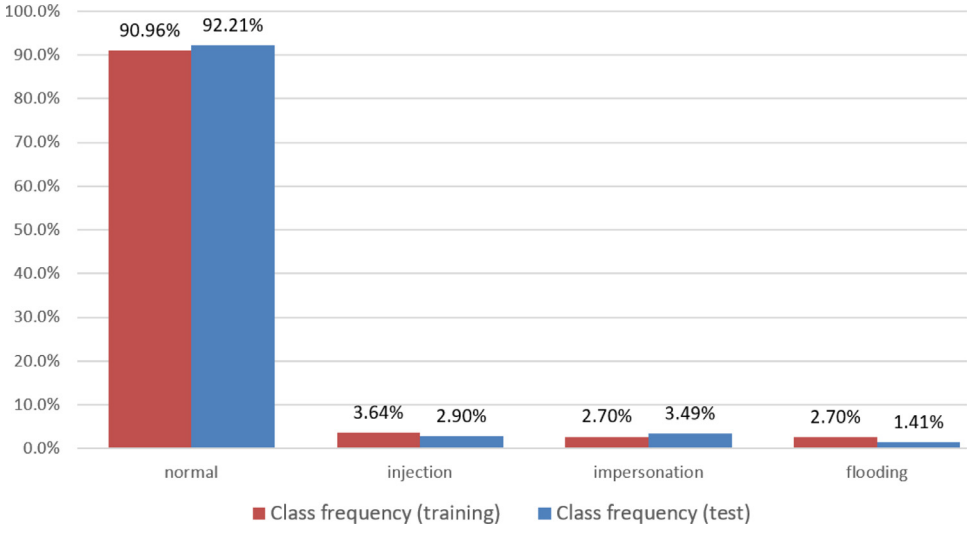


Fig. 2. Frequencies for intrusion categories for the training and test sets (AWID).

shown in Kolias et al. (2016). We have performed a [0–1] scaling to the continuous features and one-hot encoding the categorical ones.

This dataset is extremely unbalanced with over 90% normal labels. It is also newer and contains more samples than NSL-KDD. Fig. 2 shows the class distribution of the training and test AWID datasets. This dataset is more unbalanced than NSL-KDD, but, unlike NSL-KDD, the label distribution for the training and test datasets are quite similar. Therefore, the interest of experimenting together with the AWID and NSL-KDD datasets is that each offers different challenges to a classification algorithm.

3.2. Models description

This section explains the different DRL models studied in this work. There are excellent introductions to DRL (Arulkumaran et al., 2017), here we provide a brief summary. DRL is a type of reinforcement learning (RL) which uses deep learning models (e.g. NN, convolutional NN…) as function approximators for the policy and/or value functions used in RL.

RL is based in the Markov Decision Process (MDP) theory. An MDP is a tuple S, A, T, R, where S is a set of states, A is a set of actions, T is a mapping defining the transition probabilities from every state-action pair to every possible new state, and R is a reward function which associates a real value (reward) to every state-action pair. In an MDP the function T follows the Markov property, which means that the transition probability to a new state depends exclusively on the current state and action regardless of previous history. Once the MDP is defined, a policy for an MDP is a mapping from each state to an action. The objective of an MDP is to learn the optimal policy that gives the best action for every state in order to achieve a best sum of expected rewards. This optimality criterion can be further selected as a simple sum of rewards, an average or a sum of discounted rewards (when the current rewards are considered more important than future ones).

An MDP is the theoretical framework used to characterize an agent interacting with an environment in a sequential decision-making process; where the environment implements the T and R functions and the agent implements the policy. The interaction between the agent and the environment is usually discretized in a sequence of “time-steps”, in which the agent provides a new action to the environment that in turn generates a state transition and a possible new reward.

The link between the optimality criterion and the policy is usually made by defining a value function, which is an estimate of the value associated with each state. That is, an estimate of how good it is to be in a certain state considering that we will move forward with the current policy. The value function can be a V-function or a Q-function. The V-function estimate a value for each state and the Q-function estimate a value for each pair of state-actions. Both are related, being the value of the Q-function the sum of the reward for the given state-action pair plus the value of the V-function for the next state generated by the environment.

When the T and R functions of the MDP are known, we can use model-based solution techniques to obtain the optimal policy. These techniques are based on the previously defined value function plus some theoretical results: Bellman optimality equation and Generalized Policy Iteration (GPI) (Sutton & Barto, 1998). These methods are usually considered in the field of dynamic programming solutions, with two alternative implementations: policy and value iteration; the difference between them being the way to carry out the GPI mechanism. Even when the aforementioned theory and implementation alternatives are applicable to model-based solutions, they form the basis of all RL methods (model-based or model-free).

When T and R are not known, we must apply model-free solution techniques to obtain the optimal policy. In this case, there are also two options, we can try first to learn the model (T and R) and then apply the previous techniques once T and R are known, and, the other alternative is to try to learn the best policy directly without knowing first T and R. This latter scenario is the one taken by most of the RL models, and it is the one considered for this work. The problem in this scenario is that not knowing the dynamics (T and R) of the environment prevents us from using the Bellman equation, which requires knowing the transition probabilities of each state to all other possible states. There are different solutions in this case: (1) TD learning algorithms, (2) policy gradient and (3) Monte Carlo methods. TD methods are based on considering a single transition of states (from the current state to the next state under the current policy) in a slightly modified Bellman equation, instead of considering all possible transitions as required by the original Bellman equation. DQN, DDQN and actor-critic methods are based on TD learning. Policy gradient methods try to learn a policy function directly using gradient descent to optimize the expected sum of discounted rewards under a current policy. Monte Carlo methods are based in exploring the frequencies of state and action pairs and their associated sum of rewards along sampled trajectories, for example, evaluating a value function for one state will be based on the average value obtained from several trajectories starting from that state.

To achieve an optimal policy, it is important to sweep (explore) as much as possible the state-action space. The main exploration approaches are ε-greedy and action-probability based. In ε-greedy the best action is selected with probability p or a random action with probability 1-p. The exploration based in action-probability assumes that the policy (directly or indirectly) provides a probability for each of the actions, which allows a sampling process according to this probability distribution.

We can conclude that the learning of the value or policy functions is based on different types of iterative processes, where each iteration generates a function adjustment. To implement these functions there are two main alternatives: look-up tables and function approximators. A look-up table is based on a data structure that stores the value of the function for each possible combination of input values (state and/or action); meanwhile, a function approximator calculates a value for each possible combination. The chosen alternative is very important when the state and/or action spaces have a high dimensionality, with a strong impact on performance and storage capacity. All DRL methods use function approximators based on different NN variants.

3.2.1. Datasets preparation

A generic task for all models is to handle the dataset to create the mini-batches (set of samples used in a training iteration) that each specific model will use. The training dataset contains N samples of network features and associated intrusion labels with several possible values (binary or multiclass anomaly). To assimilate these elements to DRL concepts, we consider the network features as states and the label values as actions. The training is performed with mini-batches of samples formed by: (1) a state, (2) its correct label and (3) a next state.

A mini-batch will be a subset of samples drawn at random from the dataset. Each training pass is done with a different mini-batch that is updated by random sampling from the dataset.

Fig. 3 shows the structure of a mini-batch used by the DQN, DDQN and actor-critic models. In this case, a mini-batch is formed by $n+1$ random samples of the dataset. The generation process for each mini-batch is to randomly permute the dataset before the process is initiated, and then choose $n+1$ consecutive samples starting from a random index (t).

The policy gradient model forms the mini-batches in a different way. Policy gradient (Bartlett & Baxter, 2011) needs a sequence of states forming an episode (trajectory), which is the reason to build training mini-batches integrated by n trajectories of length T. Contrary to what happens when dealing with a live environment where the length of the trajectories is not fixed, in our case we can choose the length of the trajectories since we construct them by sampling from the dataset.

In Fig. 4 is presented the structure of a mini-batch used by the policy gradient model. The structure is formed in a similar way to that presented in Fig. 3, but, in this case, we create $n+1$ consecutive trajectories of length T. To ensure that the trajectories are randomly chosen, we also make a random permutation of the dataset before the construction of each mini-batch.

The dataset preparation depicted in Figs. 3 and 4 will apply to the two datasets used in this work (NSL-KDD and AWID)

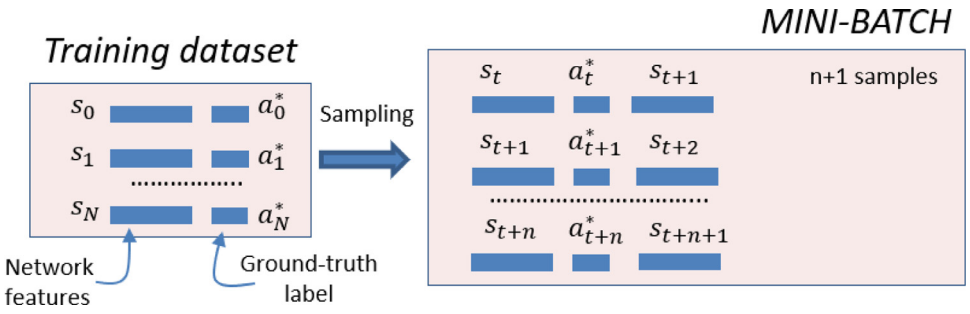


Fig. 3. Dataset preparation for the training of the DQN, DDQN and actor-critic models.

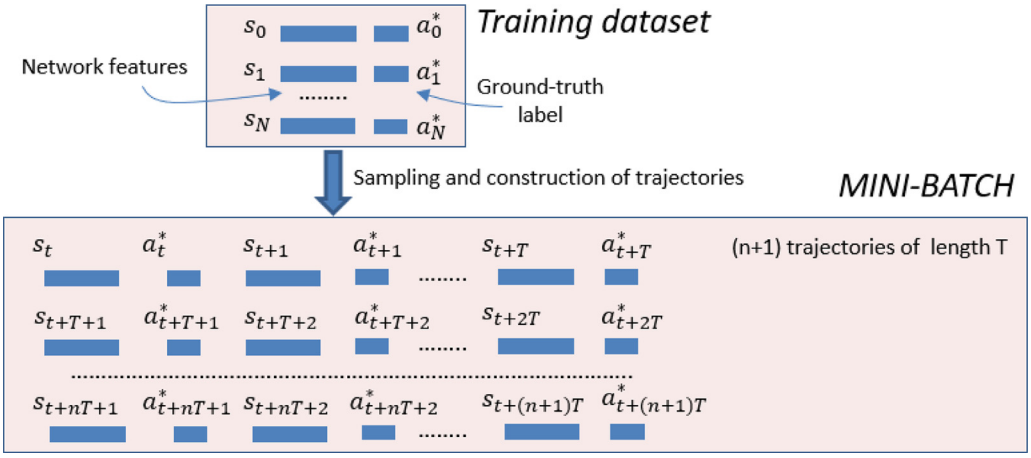


Fig. 4. Dataset preparation for the training of the policy gradient model.

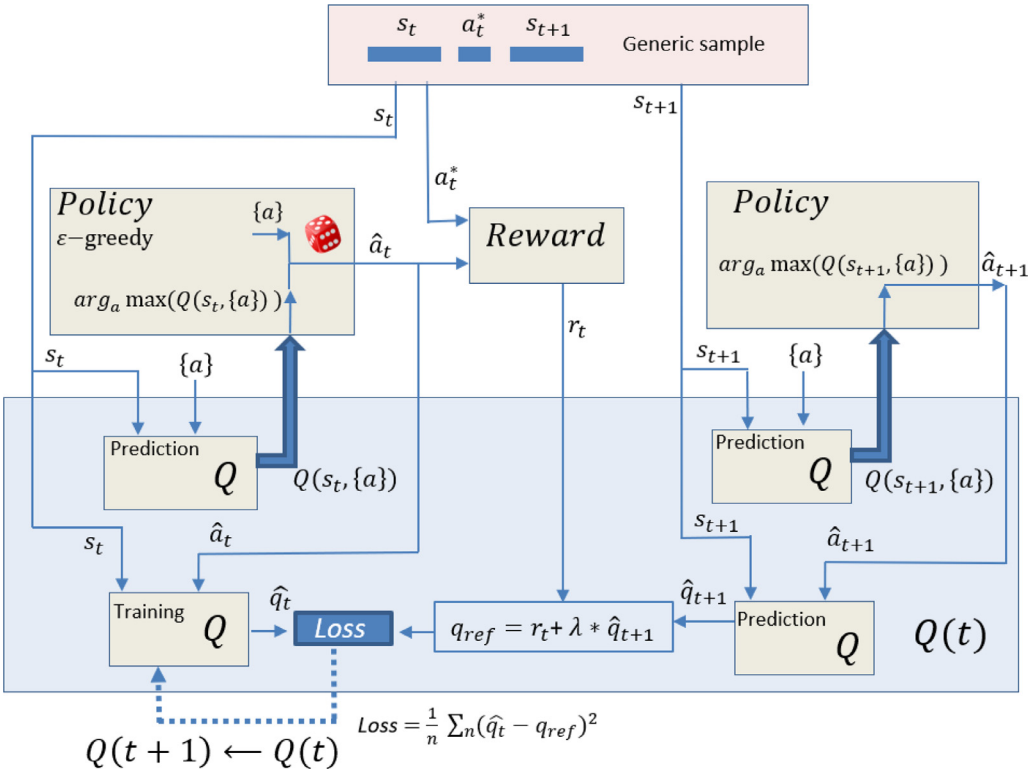


Fig. 5. Schema of the DQN model during training.

3.2.2. Models detail

This section presents in detail the different DRL models used for this work.

3.2.2.1. DQN. The algorithm employed for training DQN (Mnih et al., 2013) is depicted in Fig. 5.

The objective of the DQN model is to approximate the Q function. A Q function provides the maximum expected reward under a specific state and action, therefore, depending on a state and action pair: $Q(s,\ a)$. Once the Q function is obtained, then we can easily get the policy function which is the function that designates the action to take for each state. The policy function depends on the state and it is derived from the Q function as follows: Policy $(s) = \arg_a \max(Q(s,\ a))$. That is, it is the action that maximizes the Q-value.

The algorithm depicted in Fig. 5 starts from a generic sample formed by the current state (s_t), the ground-truth label for the current state (a_t^*) and the next state (s_{t+1}).This generic sample is part of a mini-batch of n samples. The process followed to create this generic sample from the original dataset is presented in Section 3.2.1. Each training iteration of the algorithm will process all the samples of a mini-batch, and for each iteration a new mini-batch is built following the process shown in Section 3.2.1.

A neural network (NN) is used as a function approximator for the Q function. We have used a simple NN of 3-layers, with ReLU activation for all layers, including the last one to ensure a positive Q-value. The NN training is done with a Mean Square Error loss between the Q-value estimated by the NN for the current state $(\hat{q}_t)$ and a reference value: q_{ref}, obtained by adding the current reward (r_t) to the Q-value for the next state $(\hat{q}_{t+1})$ multiplied by a discount factor (λ).

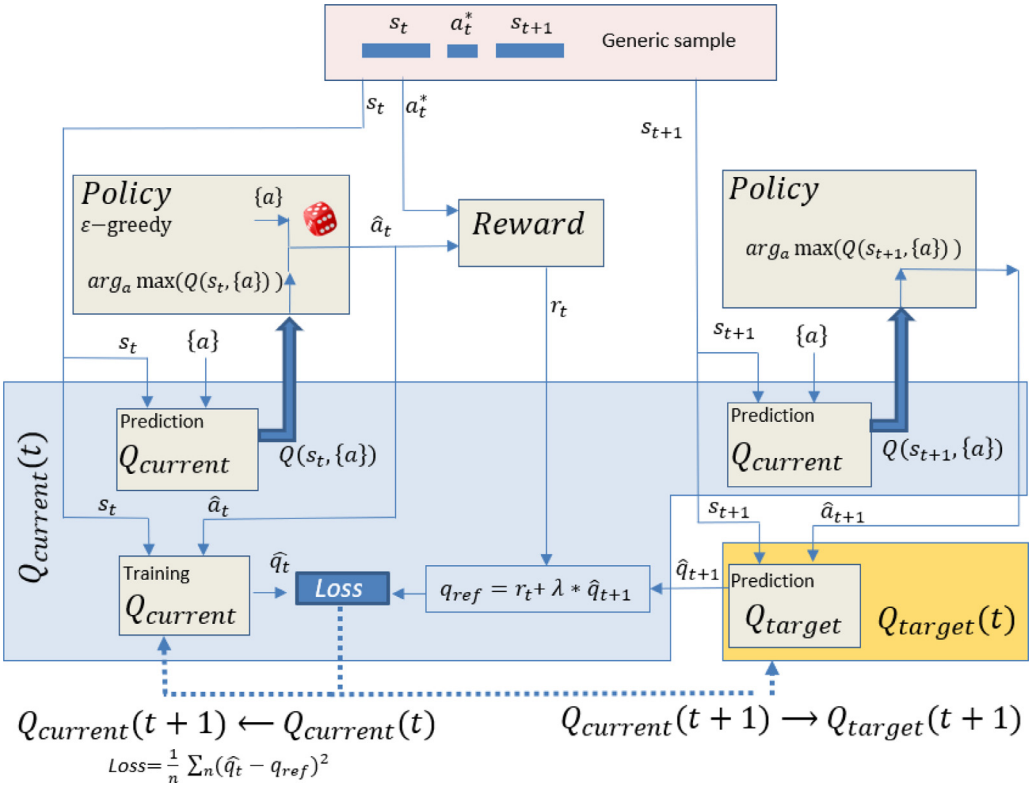


Fig. 6. Schema of the DDQN model during training.

The reward is a 1/0 reward associated, respectively, to a correct/incorrect prediction. In Fig. 5, the ground truth label value for the current state is represented by a_t^* and the predicted value by $\hat{a}_t$. When these two values are equal the reward is 1 and 0 otherwise.

To obtain the predicted value for the current state ($\hat{a}_t$) we iterate the Q function with the current state (s_t) and all possible values of the label ($\{a\}$). This iteration is represented in Fig. 5 as: $Q(s_t,\{a\})$, which is a vector of values and is shown as a thicker arrow: $Q(s_t,\{a\}) = [Q(s_t,\{a\}_0), Q(s_t,\{a\}_1),..Q(s_t,\{a\}_p)]$, where $\{a\}$ is the set of all possible actions and p is the cardinality of this set.

Then, we choose the action value which produces the maximum Q-value obtained from this iteration: $\arg_a max(Q(s_t,\{a\}))$. The action chosen is further applied to an ε-greedy algorithm which selects that value with probability p or a random action with probability 1-p. The result of this last step provides the predicted action ($\hat{a}_t$).

The predicted action for the next state ($\hat{a}_{t+1}$) is obtained in a similar way but ignoring the ε-greedy selection (right part in Fig. 5). This predicted action together with the next state (s_{t+1}) is used to obtain the Q-value for the next state ($\hat{q}_{t+1}$) which is used to calculate qref (right part in Fig. 5). The prediction of the Q-value for the next state ($\hat{q}_{t+1}$), as presented on the right part of Fig. 5, can be simplified to: $\hat{q}_{t+1} = \max_a Q(s_{t+1}, \{a\})$.

The best performance was obtained by applying a small value for the discount factor (λ). Small values of λ give more importance to learning the current reward, regardless of the succession of future rewards. This makes sense, considering that (1) the next state is uncorrelated with the present state, and (2) we are actually implementing a classifier disguised as a DRL algorithm, being the real goal to make a correct prediction for the current state (current reward).

In order to make a faster algorithm, our final model used for DQN was slightly modified with respect to the one shown in Fig. 5. The network implementation for the Q function consists of two input vectors (state and action) and one scalar as output (Q-value). This architecture requires iterating the network with all the action values to recognize the value that maximizes the Q function. This iteration slows down the process. As a solution, in Mnih et al. (2013) is presented an alternative network architecture with a single input vector (state) and one output vector with the same length as the number of action values (i.e. number of intrusion classes, since they are one-hot encoded). In this way, each element of the output vector has the Q-value of its associated action value through a single iteration.

Besides the 1/0 reward we tested other distance functions between the predicted and true labels (actions), such as cross-entropy and hinge function. Of all the functions analyzed, the 1/0 reward is the one that has provided the best results, despite its simplicity.

Once the training of the model is completed, the NN implementing the Q function is used for prediction. For a particular state, the Q function will provide a Q-value for each of the possible actions for that state. The predicted action is the one that maximizes the Q-value (no ε-greedy applied for prediction).

The model was trained for 10 epochs, where an epoch is a number of iterations enough to cover the complete dataset.

3.2.2.2. DDQN. The DDQN (Van Hasselt et al., 2015) model is presented in Fig. 6, where the algorithm applied to the training phase is shown.

The DDQN model is very similar to DQN. The only difference being that two NNs are employed in DDQN: one implements a current Q function while the other one implements a target Q function. The target Q function is a copy of the current Q function, but with a delayed synchronization; that is, the copy is made after a certain number of training iterations. The target Q function is used to calculate the Q-value for the next state ($\hat{q}_{t+1}$). The intention of this additional Q function (target Q function) is to avoid the moving target effect when doing gradient descent over $(\hat{q}_t - q_{ref})^2$,

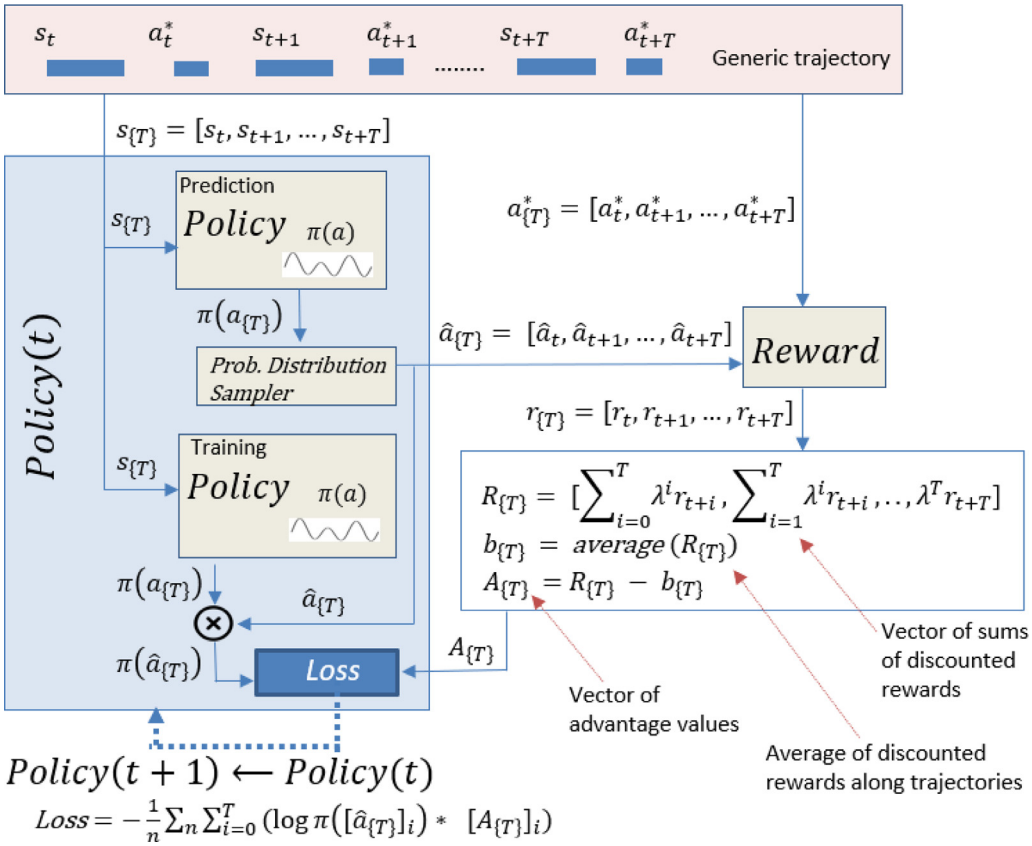


Fig. 7. Schema of the policy gradient model during training.

avoiding the recursive dependence of q_{ref} on the training network (Q current) (Van Hasselt et al., 2015).

Besides the addition of the target Q function, the algorithm depicted in Fig. 6 is similar to Fig. 5, and all the mechanisms explained in Section 3.2.2.1 are applicable here. The hyperparameters used were also similar to the ones for the DQN model.

3.2.2.3. Policy gradient. The diagram in Fig. 7 shows the details of the training process of the policy gradient model (Bartlett & Baxter, 2011) employed for this work.

Policy gradient is based on training a policy function, which designates the action that must be taken for each possible state. The policy function is approximated by a simple NN that has been implemented with a few layers and ReLU activation for all layers except the last layer which has a softmax activation that provides a probability distribution of the actions ($\pi(a)$).

The algorithm depicted in Fig. 7 uses a generic trajectory consisting of a sequence of T pairs formed by a state (s_t), and its associated ground-truth label (a_t^*). This generic trajectory is part of a mini-batch of n trajectories. The process followed to create this generic trajectory from the original dataset is presented in Section 3.2.1. Each training iteration of the algorithm will process all the trajectories of a mini-batch, and for each iteration a new mini-batch is built following the process shown in Section 3.2.1.

The algorithm begins by predicting the actions using the states and the policy function. The action prediction is done for all the states in a trajectory ($s_{\{T\}}$), producing a sequence of predicted actions: $\hat{a}_{\{T\}}$. These predicted actions are obtained by sampling on the probability distribution of the actions ($\pi(a_{\{T\}})$) provided by the policy function. This is represented as the "Prob. Distribution Sampler" in Fig. 7.

We use the symbol $\{T\}$ to represent a sequence along the time-steps of one trajectory. When we use this symbol we can have a sequence of scalar values, as in $r_{\{T\}}$ or a sequence of vectors such as $\pi(a_{\{T\}})$ or $\hat{a}_{\{T\}}$, since in the latter case, $\pi(a)$ is a vector of probabilities for each possible action under the current policy, and $\hat{a}_t$ is a one-hot encoded vector with a 1 assigned to the picked action, therefore their extension to a sequence produces a sequence of vectors.

Similarly to Section 3.2.2.1, the reward function generates a 1/0 reward, but, in this case, it is applied to a whole sequence of predicted actions ($\hat{a}_{\{T\}}$) and ground-truth actions ($a^*_{\{T\}}$) in a trajectory. The reward sequence ($r_{\{T\}}$) obtained is transformed into a vector of sums of discounted rewards ($R_{\{T\}}$).

$R_{\{T\}}$ is calculated with the following expression:

$$R_{\{T\}} = \left[\sum_{i=0}^{T}\lambda^i r_{t+i}, \sum_{i=1}^{T}\lambda^i r_{t+i}, .., \sum_{i=T}^{T}\lambda^i r_{t+i}\right] = \left[\sum_{i=0}^{T}\lambda^i r_{t+i}, \sum_{i=1}^{T}\lambda^i r_{t+i}, .., \lambda^T r_{t+T}\right]$$

That is, each term of $R_{\{T\}}$ corresponds to a decreasing sum of terms of consecutive discounted rewards.

From the vector of sums of discounted rewards ($R_{\{T\}}$) we subtract the average of discounted rewards along trajectories ($b_{\{T\}}$), resulting in the vector of advantage values ($A_{\{T\}}$). The vector $b_{\{T\}}$ is also called a baseline. The advantage values provide an estimate on how much the expected return for a certain element of the trajectory (s_t) is better than the average expected return (that is the reason to subtract the baseline from $R_{\{T\}}$).

The scalar product between the sequence of vectors $\pi(a_{\{T\}})$ and $\hat{a}_{\{T\}}$, extracts the probability of the picked action for each time-step

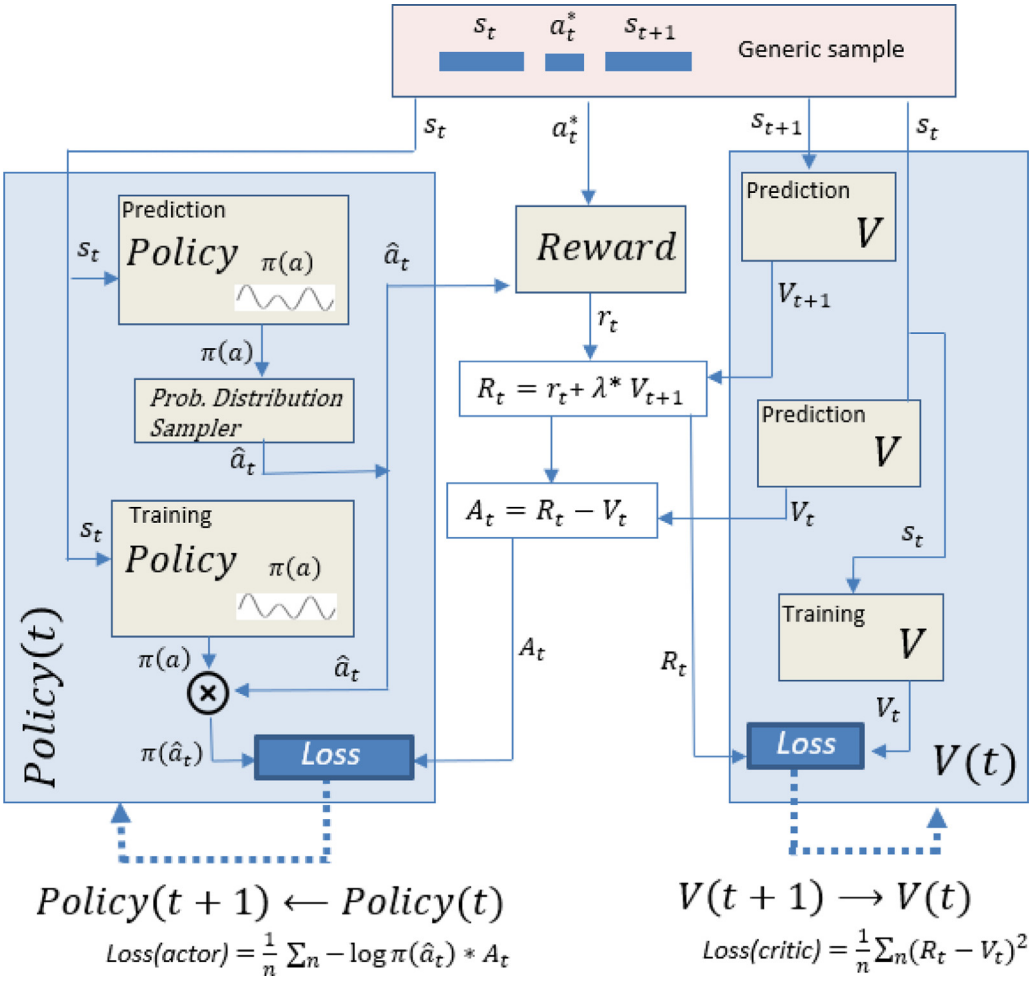


Fig. 8. Schema of the Actor-Critic model during training.

$(\pi(\hat{a}_{\{T\}}))$, since $\hat{a}_t$ is a one-hot encoded vector. A product symbol inside a circle is used in Fig. 7 to represent the scalar product.

The loss used to train the NN that approximates the policy function, is a kind of log-loss function formed by the sum along trajectories of the log of the probability of the picked action for a certain element of the trajectory $(\log \pi([\hat{a}_{\{T\}}]_i))$ multiplied by its corresponding advantage value $([A_{\{T\}}]_i)$. The complete process is depicted in Fig. 7.

When the training is finished, for prediction we use the NN implementing the policy function. For a particular state, the policy function will provide a distribution of probabilities for the actions. In this prediction case, we just choose the action with the highest probability (without sampling).

3.2.2.4. Actor-critic. The schema for the training of the actor-critic (Grondman et al., 2012) model is shown in Fig. 8. For the actor-critic model we use two function approximators: one for the policy function, another one for the Value function. The Value function provides the expected sum of rewards starting from a designated state, and the policy function indicates the action to take under a given state. The two function approximators are implemented with two NNs of 3-layers and ReLU activation for all layers except the last layer of the policy function with softmax activation. The softmax activation provides the probability distribution of the actions.

By addressing both functions simultaneously, they both help each other in the training process. Under a given current state (s_t), the policy function is used to estimate the ground truth action (a_t^*). The policy function provides the probability distribution $(\pi(a))$ of the actions under a given state. This probability distribution is used in a sampling process to choose the preferred action $(\hat{a}_t)$. The selected action $(\hat{a}_t)$ together with the ground-truth action (a_t^*) is used by the reward function to provide a 1/0 reward associated to a correct/incorrect action selection. If the picked action matches the ground-truth label, a reward of 1 is given, otherwise the reward is 0. This is the current reward (r_t).

The algorithm depicted in Fig. 8 starts from a generic sample formed by the current state (s_t), the ground-truth label for the current state (a_t^*) and the next state(s_{t+1}).This generic sample is part of a mini-batch of n samples. The process followed to create this generic sample from the original dataset is presented in Section 3.2.1.

The previously mentioned current reward (r_t) is used to make an estimation of the value function (R_t) associated with the current state. This estimation is employed to train the value function (with a mean square error loss). The difference between this estimation of the value function (R_t) and the actual value function (V_t), obtained from the NN approximator, is an estimation of the advantage value (A_t) for the current state. This advantage value is used, similarly to policy gradient, to train the policy function. The training is done with a loss function formed by the log of the probability of the picked action $(\hat{a}_t)$ under the probability distribution given by the policy function $(\log \pi(\hat{a}_t))$, and weighted by the corresponding advantage value (A_t).

Similar to policy gradient (Section 3.2.2.3) we use a scalar product between the selected action ($\hat{a}_t$) and the probability distribution for all actions ($\pi(a)$) to extract the probability of the picked action ($\pi(\hat{a}_t)$), since $\hat{a}_t$ is a one-hot encoded vector. A product symbol inside a circle is used in Fig. 8 to represent the scalar product.

The complete model is trained by consecutively iterating the value and policy functions with Stochastic Gradient Descent (SGD) to decrease the corresponding loss for each function: a log-loss for the policy and a quadratic loss for the value function, respectively.

The data preparation for this model is similar to DQN. It does not require adapting the data to trajectories of states and actions, as needed for the policy gradient model.

For prediction, in a similar way to the policy gradient model, we use the NN that implements the policy function, simply by selecting the action with the highest probability (without sampling).

Our implementation of the model follows conceptually the schema presented in Fig. 8, however for our final model we apply some minor tricks to reduce the training time. For example, we gather in a single NN the value and policy functions, implementing an NN with the state as input and k different outputs for the value and action, respectively; where k is the number of intrusion labels to predict (one hot encoded). Therefore, we can perform the training of both functions in a single iteration, employing a loss function formed by the sum of the losses of the two original functions.

4. Results

This section provides a comparison of the DRL models with a series of alternative common ML models applied to the AWID and NSL-KDD datasets. The alternative ML models are: Support Vector Machine (SVM), Logistic Regression, Naive Bayes (NB), K-Nearest Neighbors (KNN), Random Forest, Gradient Boosting Machine (GBM), AdaBoost, MLP and Convolutional Neural Network (CNN). All these models are compared with the DRL models: DQN, DDQN, policy gradient and actor-critic.

The results are obtained with the full NSL-KDD and AWID test sets (Section 3.1), without any sampling or preparation of an alternative test set taken from the training data. This is an important difference when comparing results with other works, as the unbalanced and difficult prediction structure of the original datasets can be significantly altered by choosing a different test set.

		Test results			
		Accuracy	F1	Precision	Recall
Logistic Regression	Logistic Regression	0.7068	0.6807	0.8955	0.5491
SVM	SVM, Linear Kernel	0.7740	0.7718	0.9073	0.6715
	SVM, RBF Kernel	0.8799	0.8927	0.9081	0.8779
KNN	K-Nearest Neighbors	0.7808	0.7769	0.9233	0.6706
Random Forest	Random Forest	0.7472	0.7211	0.9688	0.5743
GBM	Gradient Tree Boosting	0.7761	0.7612	0.9690	0.6267
Naive Bayes	Gaussian, only continous features	0.5490	0.7018	0.5627	0.9324
	Bernoulli, only discrete features	0.7942	0.7979	0.9046	0.7138
	Bernoulli, all features, quantizing the continous features	0.8019	0.7967	0.9583	0.6818
AdaBoost	Adaboost with Trees	0.7606	0.7403	0.9683	0.5992
	Adaboost with Naive Bayes	0.5265	0.3181	0.8824	0.1940
Neural Network	Neural Net (MLP)	0.7966	0.7881	0.9679	0.6647
CNN	CNN-1D	0.7875	0.7633	0.8094	0.7875
Reinforcement Learning	DQN	0.8787	0.8935	0.8933	0.8937
	DDQN	0.8978	0.9120	0.8944	0.9303
	Policy Gradient	0.7873	0.7909	0.8980	0.7067
	Actor Critic	0.8078	0.8111	0.9203	0.7251

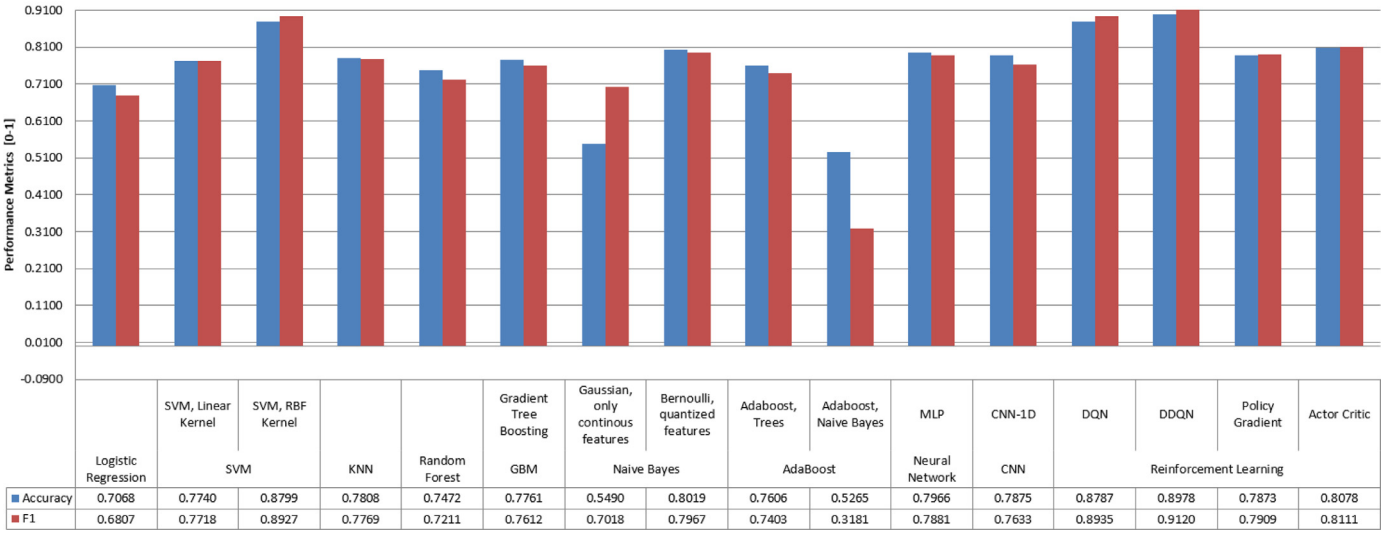


Fig. 9. Performance scores for all models (NSL-KDD dataset).

		Test results	
		Accuracy	F1
λ = 0.01	DQN	0.8787	0.8935
	DDQN	0.8978	0.9120
	Policy Gradient	0.7873	0.7909
	Actor Critic	0.8078	0.8111
λ = 0.99	DQN	0.4816	0.2800
	DDQN	0.6197	0.5400
	Policy Gradient	0.7710	0.7556
	Actor Critic	0.7842	0.7863

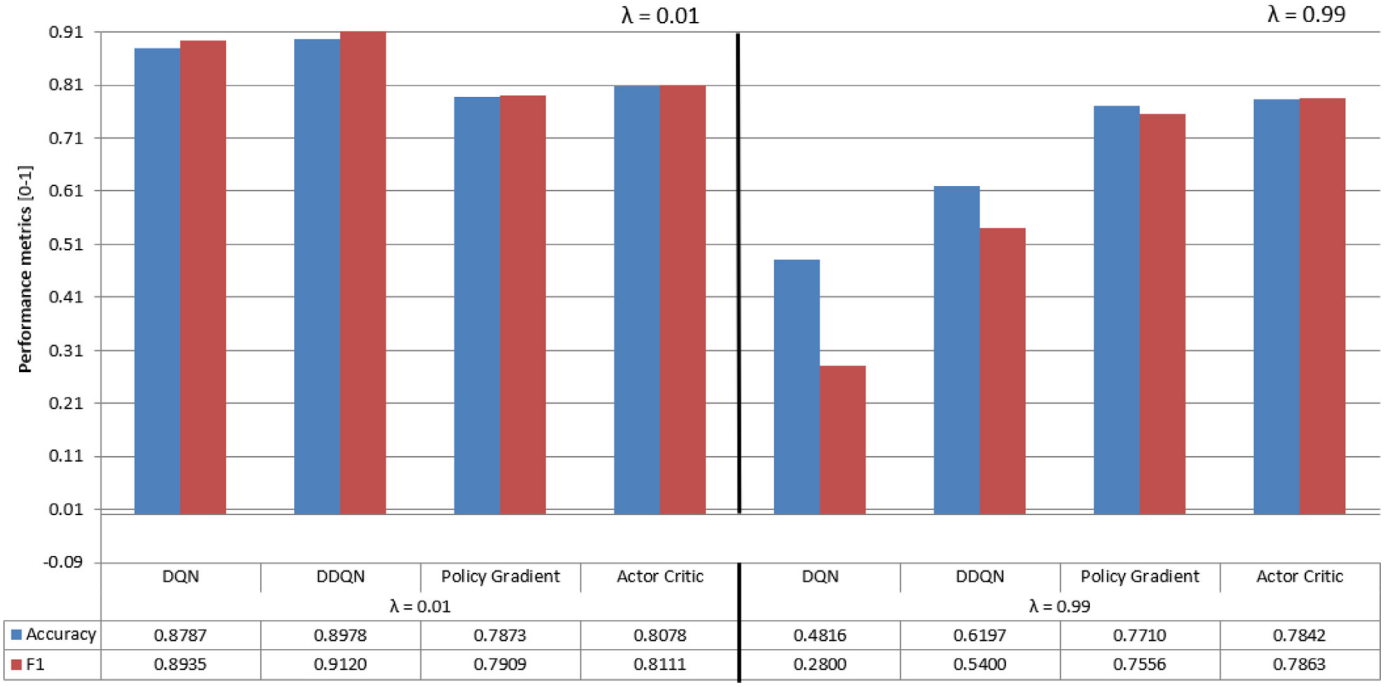


Fig. 10. Comparison of performance scores for different discount factors (NSL-KDD dataset).

The performance metrics used have been: F1-score, accuracy, precision and recall (Bhuyan et al., 2014). Due to the unbalanced nature of the datasets (mainly AWID) we will provide more importance to the F1 score which is more suitable in case of imbalance datasets.

To present the results for the two datasets, we first provide the results of the experiments with NSL-KDD dataset followed by the AWID dataset.

4.1. Results for the NSL-KDD dataset

One of the main contributions of this work is to show the adequacy of the DRL models for intrusion detection in networking. In Fig. 9 we present the results for the four DRL models that have been studied (DQN, DDQN, Policy Gradient and Actor Critic) when applied to the NSL-KDD dataset. Fig. 9 presents the results in two parts. The upper part presents the raw data in a color-coded way, where the greenest is associated with a better value and the redder with a worse value (comparison of values is applied column-wise). And, the lower part presents only the accuracy and F1 scores (the most significant scores) in a chart; in this case, a variant of Naive Bayes has been eliminated from the graph to make it less cluttered considering the scarce importance of this model in terms of results.

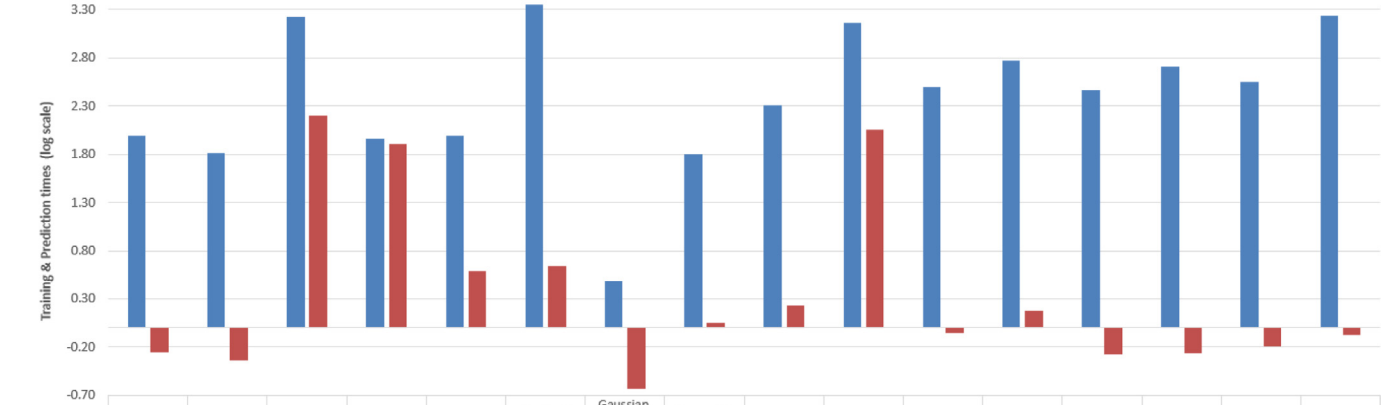
The main results for alternative ML models have been obtained from Lopez-Martin et al. (2019) where the NSL-KDD dataset was applied to several ML models in order to score them. However, for this study we have expanded the results for Adaboost and incorporate NB with several variants: a Gaussian NB using only the continuous features, a Bernoulli NB using only the discrete features and finally a Bernoulli NB using all features. In order to apply the Bernoulli NB to the continuous features, we needed to transform them from continuous to discrete intervals (quantization).

In addition to the detection scores for the DRL models, Fig. 9 shows the performance metrics for all models. We can observe how the DDQN and DQN models produce the best results (considering F1, accuracy and recall), followed by SVM (with RBF kernel) and actor-critic models.

The NSL-KDD dataset presents many challenges for a classifier, with the different composition of the training and test sets being one of the most important. We can also observe in Fig. 9 how DDQN stands out in the Recall metric. This metric is very important to guarantee a minimum number of false negatives (samples with an intrusion that are predicted to be normal), which is the main performance metric for an intrusion detection system that tries to identify as many intrusions as possible, considering all of them as critical.

As noted in Section 3.2, the discount factor (λ) considered for the DRL algorithms can have a significant influence on the results.

		Training time (sec)	Prediction time (sec)
Logistic Regression	Multinomial	97.37	0.55
SVM	SVM, Linear Kernel	65.06	0.46
	SVM, RBF Kernel	1696.16	158.65
KNN	KNN	91.53	80.30
Random Forest	Random Forest	97.31	3.87
GBM	Gradient Tree Boosting	2242.14	4.39
Naive Bayes	Gaussian, only continous features	3.04	0.23
	Bernoulli, only discrete features	5.52	0.36
	Bernoulli, quantized features	63.89	1.12
AdaBoost	Adaboost, Naive Bayes	1465.73	113.22
Neural Network	Neural Net (MLP)	314.74	0.89
CNN	CNN-1D	590.58	1.52
Reinforcement Learning	DQN	290.50	0.54
	DDQN	507.01	0.55
	Policy Gradient	352.48	0.65
	Actor Critic	1725.17	0.83



	Logistic Regression	SVM, Linear Kernel	SVM, RBF Kernel	KNN	Random Forest	Gradient Tree Boosting	Gaussian, only continous features	Bernoulli, quantized features	Adaboost, Trees	Adaboost, Naive Bayes	MLP	CNN-1D	DQN	DDQN	Policy Gradient	Actor Critic
		SVM		KNN	Random Forest	GBM	Naive Bayes		AdaBoost		Neural Network	CNN	Reinforcement Learning			
Training time log scale (sec)	1.99	1.81	3.23	1.96	1.99	3.35	0.48	1.81	2.30	3.17	2.50	2.77	2.46	2.71	2.55	3.24
Prediction time log scale (sec)	-0.26	-0.33	2.20	1.90	0.59	0.64	-0.63	0.05	0.23	2.05	-0.05	0.18	-0.27	-0.26	-0.19	-0.08

Fig. 11. Computation times for training and prediction of models (NSL-KDD dataset).

To study this influence, Fig. 10 shows the impact of different values for the discount factor used in the DRL models. The impact is critical for DQN and DDQN and less important for policy gradient and actor-critic models. We obtain better results for very low values of the discount factor, as explained in Section 3.2.2.1. This is clearer for the DQN and DDQN models than for the policy gradient model, since the latter optimizes the network parameters based on a sequence (episode) of states/actions, implicitly performing an optimization on an averaged sum of rewards, whereas the DQN and DDQN models are based on single state and action pairs.

Fig. 10 also presents the data in two sections, with the upper section presenting the raw data with a color-code similar to that of Fig. 9, and the lower section showing the same data in a chart format. This same data presentation structure is used for Figs. 11 and 12.

As presented earlier, one of the advantages of the DRL models is that once trained, the resulting policy function, which provides the correct intrusion label (action) for a specific state (intrusion features), is actually a simple neural network which can be extremely fast for the inference (prediction) stage and, therefore, suitable to be used in an industrial production environment. This behavior can be checked in Fig. 11, which gives the prediction and training times for the models. We can appreciate how the computation times for prediction for the DRL models are really very small compared to the second model with better results (SVM-RBF). For training times, the values are also smaller for the DRL models, with the exception of the actor-critic model, which requires a considerable amount of training time. As expected, the best training times are obtained for the linear models (logistic regression and linear SVM) together with Random Forest and Naive Bayes.

In Fig. 11, the color-coded raw data table (upper part) is complemented with an associated chart (lower part) of the training and prediction times in logarithmic scale. Considering the large range of values, the logarithmic scale allows a more suitable view

		Test results			
		Accuracy	F1	Precision	Recall
Boosting	AdaBoost	0.9220	0.8850	0.8500	0.9220
Decision Tree	J48	0.9620	0.9480	0.9620	0.9630
Frequency Table based	Hyper pipes	0.9223	0.8850	0.8790	0.9220
	Naive Bayes	0.9055	0.9090	0.9170	0.9060
	ZeroR	0.9220	0.8850	0.8500	0.9220
	OneR	0.9457	0.9220	0.9000	0.9460
Random Forest	Random Forest	0.9582	0.9440	0.9590	0.9580
Neural Network	MLP	0.9470	0.9256	0.9174	0.9473
Reinforcement Learning	DQN	0.9541	0.9372	0.9244	0.9541
	DDQN	0.9570	0.9394	0.9235	0.9570
	Policy Gradient	0.9221	0.8847	0.8502	0.9221
	Actor Critic	0.9221	0.8847	0.8502	0.9221

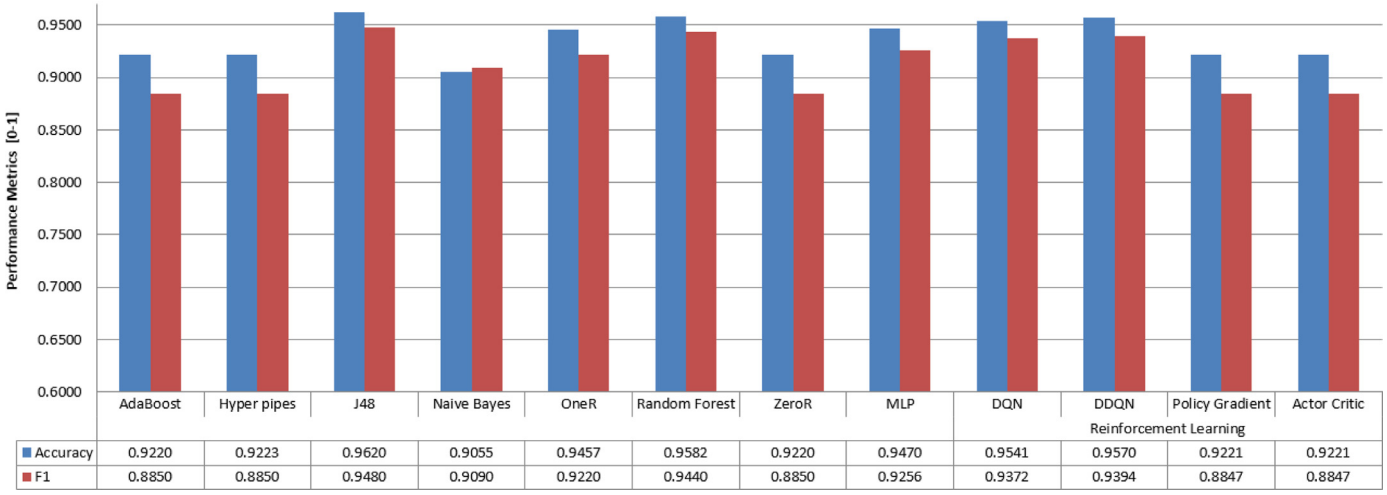


Fig. 12. Performance scores for all models (AWID dataset).

of the data. In the chart of Fig. 11, the smallest values (positive or negative) correspond to a better value in terms of training or prediction times.

4.2. Results for the AWID dataset

AWID is an extremely unbalanced dataset, useful to test the performance of an intrusion detector (Section 3.1.2). Fig. 12 presents the performance results of the different DRL models studied in this work together with a wide range of alternative ML models. The results for the alternative ML models have been extracted from the original work of the research team that created the data set (Kolias et al., 2016), where in addition to an analysis of the reasons for its creation and a detailed description of its composition, it is presented a complete study of the expected classification results using different ML models.

For the AWID dataset we have a multi-class classification problem. In a multi-class classification, the results can be provided in two possible ways: aggregated and one vs. rest.

In the case of one vs. rest, we consider each particular class (label) against all other classes, reducing it to a sequence of binary classifications (one for each particular class). In the case of aggregated results, we provide a single result which is a summary (average) for all classes. There are different possibilities for the aggregation (micro, macro, samples, weighted), with different averaging methods.

The performance metrics provided in Fig. 12 are aggregated metrics employing a weighted average for the F1, precision and recall as shown in Pedregosa et al. (2011). We can observe how the DDQN model presents excellent results for the Accuracy, F1 and Recall metrics. For this dataset, the Random Forest and Decision Tree (J48) models obtain the best results. It is important to mention that DDQN excels again in the Recall metric for this dataset, which, as mentioned for the results of NSL-KDD (Section 4.1), is a critical metric for an intrusion detection algorithm that attempts to reduce false negatives (intrusions that are not detected).

Table 1
One vs. rest performance metrics for the DDQN model (AWID dataset).

	Metric			
Label	Accuracy	F1	Precision	Recall
flooding	0.9939	0.7403	0.9219	0.6185
impersonation	0.9651	0.0000	0.0000	0.0000
injection	0.9980	0.9662	0.9346	0.9999
normal	0.9570	0.9772	0.9581	0.9970

Considering the one vs. rest metrics for the 4 class values of the AWID dataset, the results are provided in Table 1. In this table, each row corresponds to the scores obtained when considering a binary classification between a label (the one associated with the row) and the rest of the labels. Even considering the poor results for one of the class values: impersonation, the rest presents very good results. It is important to keep in mind the strong unbalanced nature of the dataset, and how easy is for a classifier, under these conditions, to adopt the majority class as the unique classification result. This only occurs for one of the class values (impersonation), as can be seen in detail in Table 2 that shows the confusion matrix (Bhuyan et al., 2014) for the DDQN model applied to the AWID dataset with 4 class values. The number of true positives

Table 2
Confusion matrix for the DDQN model (AWID dataset).

		Predicted			
		flooding	impersonation	injection	normal
Real	flooding	5008	0	0	3089
	Impersonation	7	0	0	20072
	injection	0	0	16681	1
	normal	417	1	1167	529199

in Table 2 is high for all class values except for the impersonation attack, with no true positives (that is the reason for the zero value of F1, precision and recall, in Table 1) and, where almost all predictions have fallen within the normal value.

4.3. Summary of results

The DDQN model (which is the DRL model that presents best results) has a comparable detection performance (accuracy, F1, precision and recall), and, in some cases, better than alternative SOTA ML models (e.g. Random Forest, Decision Trees, SVM, Naïve Bayes…).

The results were obtained with two different IDS datasets, and in both cases our model based on DDQN has been in the set of the best solution models, while the alternative SOTA model, for each dataset, has been very different in each case (SVM and Decision Trees for NSL-KDD and AWID respectively). The conclusion is that our model provides a more robust solution in different scenarios.

It is particularly important to mention the excellent results of DDQN in the recall metric, which is crucial for an intrusion detection algorithm, since we want to reduce at a minimum the number of false negatives.

In addition to demonstrating that DRL models can be applied to intrusion detection problems with a labeled dataset, the classifiers obtained (once they are trained) are simple and fast neural networks which are suitable for distributed high-performance computing environments (e.g. Tensorflow) (Abadi et al., 2016). In particular, the DDQN model, at test time, provides much smaller prediction times than the best SOTA models studied for this work.

Considering the results presented in Fig. 10, we can conclude that the application of DRL models to a scenario with a labelled dataset depends to a large extent on the choice of the value of the discount factor. This is an unexpected and significant discovery, since it seems that the fact of not interacting with a live environment (which means that the feedback loop caused by the impact of the actions on the environment is broken), we must be more conservative in each update of our policy function, making the convergence slower but more stable. This effect is particularly important for the DQN and DDQN models due to the reasons provided in Section 4.1.

The models presented in this work have been developed in python with the scikit-learn package (Pedregosa et al., 2011), with Tensorflow for all models based on neural networks (MLP, linear-SVM, CNN, DRL and logistic regression).

5. Conclusion

As a summary, the contributions of the paper are: (1) New algorithm that improves the results of intrusion detection compared to the existing techniques of machine learning and deep learning. (2) Intrusion detection algorithm based on an extremely simple and fast policy network, especially suitable for demanding applications in modern data networks that require a rapid response. (3) The resulting model is suitable for on-line learning, which is necessary for data networks with changing environments. (4) Application of DRL for supervised learning. (5) The optimization process is driven by a rewards function that is not required to be differentiable, which makes it more flexible an applicable to all kind of problems.

We provide a comparison study of four DRL algorithms (DQN, DDQN, Policy gradient and actor-critic) and how they can be applied to a dataset labeled with intrusions instead of interacting with a live network environment. An additional analysis is provided comparing these algorithms with several alternative machine learning models, considering three main features: (1) prediction scores, (2) training and (3) prediction times, and using two different intrusion detection datasets (NSL-KDD and AWID) to facilitate the generalization of the results.

The best DRL algorithm (DDQN) has a detection performance (measured by several performance metrics: accuracy, F1, precision and recall) comparable, and, in some cases, better than a full range of SOTA ML models (e.g. Random Forest, Decision Trees, SVM, Naive Bayes…). In addition, DDQN, and in general DRL methods, present and important advantage in terms of significantly reduced prediction times, which makes them very suitable for on-line detection and new highly demanding network services (e.g. IoT networks).

In addition to the application of a DRL framework to a dataset of logged features and associated class labels (instead of a live environment capable of responding in real time to the actions of the algorithm), another important contribution of this work is to show the importance of the discount factor parameter that regulates the speed of convergence of the algorithm, being especially important to have a small value for this parameter for the convergence of the DQN and DDQN algorithms, under the restrictions imposed by this work. Another contribution of this work is to show the necessary data preparation that is required to apply the DRL models to a labeled dataset, and to propose a way of doing this preparation considering the specificities of the different models. Finally, our research on reward functions (distance between the predicted and true labels) has produced the surprising conclusion that the simple 1/0 reward function produces better results than more sophisticated alternatives, e.g. cross-entropy and categorical hinge (Section 3.2.2.1).

As future work, we plan to investigate the application of new DRL algorithms, specifically the application of multi-agent and adversarial models for DRL which can be applicable to intrusion detection problems (Busoniu, Babuska, & De Schutter, 2010; Pinto et al., 2017).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.eswa.2019.112963.

Credit authorship contribution statement

Manuel Lopez-Martin: Investigation, Formal analysis, Software, Writing - original draft. **Belen Carro:** Supervision, Validation, Writing - review & editing. **Antonio Sanchez-Esguevillas:** Supervision, Resources, Writing - review & editing.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467v2 [cs.DC].

Abdulhammed, R., Faezipour, M., Abuzneid, A., & Alessa, A. (2018). Effective features selection and machine learning classifiers for improved wireless intrusion detection. In *2018 international symposium on networks, computers and communications (ISNCC), Rome* (pp. 1–6). doi:10.1109/ISNCC.2018.8530969.

Akazaki, T., Liu, S., Yamagata, Y., Duan, Y., & Hao, J. (2018). Falsification of cyber-physical systems using deep reinforcement learning. In *In international symposium on formal methods* (pp. 456–465). Cham.: Springer. doi:10.1007/978-3-319-95582-7_27.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine, 34*(6), 26–38. doi:10.1109/MSP.2017.2743240.

Bartlett, P. L., & Baxter, J. (2011). Infinite-horizon policy-gradient estimation. arXiv:1106.0665 [cs.AI].

Berman, D. S., Buczak, A. L., Chavis, J. S., & Corbett, C. L. (2019). A survey of deep learning methods for cyber security. *Information, 10*(4), 122. doi:10.3390/info10040122.

Bhattacharjee, P. S., Fujail, A. K. Md, & Begum, S. A. (2017). A comparison of intrusion detection by K-means and fuzzy C-means clustering algorithm over the NSL-KDD dataset. In *2017 IEEE international conference on computational intelligence and computing research (ICCIC), Coimbatore* (pp. 1–6). doi:10.1109/ICCIC.2017.8524401.

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network anomaly detection: Methods, systems and tools. IEEE communications surveys & tutorials. *IEEE, 16*, 303–336 Piscataway, NJ, USA. doi:10.1109/SURV.2013.052213.00046.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., et al. (2018). A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *Journal of Internet Services and Applications, 9*, 16. doi:10.1186/s13174-018-0087-2.

Busoniu, L., Babuska, R., & De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. In D. Srinivasan, & L. C Jain (Eds.). *Innovations in multi-agent systems and applications - 1. Studies in computational intelligence*: 310. BerlinHeidelberg: Springer. doi:10.1007/978-3-642-14435-6_7.

Caminero, G., Lopez-Martin, M., & Carro, B. (2019). Adversarial environment reinforcement learning algorithm for intrusion detection. *Computer Networks., 159*, 96–109. doi:10.1016/j.comnet.2019.05.013.

Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. arXiv:1901.03407 [cs.LG].

Chen, Z., Yeo, C. K., Lee, B. S, & Lau, C. T. (2018). Autoencoder-based network anomaly detection. In *2018 Wireless telecommunications symposium (WTS), Phoenix, AZ* (pp. 1–5). doi:10.1109/WTS.2018.8363930.

da Costa, K. A. P., Papa, J. P., de Oliveira-Lisboa, C., Munoz, R., & de Albuquerque, V. H. C. (2019). Internet of things: A survey on machine learning-based intrusion detection approaches. *Computer Networks., 151*, 147–157. doi:10.1016/j.comnet.2019.01.023.

Deokar, B., & Hazarnis, A. (2012). Intrusion detection system using log files and reinforcement learning. *International Journal of Computer Applications, 45*(19), 28–35.

Feng, M., & Xu, H. (2017). Deep reinforcement learning based optimal defense for cyber-physical system in presence of unknown cyber-attack. *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1–8. doi:10.1109/SSCI.2017.8285298.

Francois-Lavet V., Henderson P., Islam R., Bellemare M. G., & Pineau J. (2018). An introduction to deep reinforcement learning. arXiv:1811.12560v2 [cs.LG].

Goodfellow, I., Shlens, J., & Ch, Szegedy. (2015). Explaining and harnessing adversarial examples. arXiv:1412.6572 [stat.ML].

Grondman, I., Busoniu, L., Lopes, G. A. D., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42*(6), 1291–1307. doi:10.1109/TSMCC.2012.2218595.

Hussain, F., Hussain R., Hassan S. A., & Hossain E. (2019). Machine learning in IoT security: Current solutions and future challenges. arXiv:1904.05735v1 [cs.CR].

Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016). A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI international conference on bio-inspired information and communications technologies (formerly BIONETICS)(BICT'15), ICST (Institute for computer sciences, social-informatics and telecommunications engineering), Belgium*. doi:10.4108/eai.3-12-2015.2262516.

Kim, K., & Aminanto, M. E. (2017). Deep learning in intrusion detection perspective: Overview and further challenges. In *2017 International workshop on big data and information security (IWBIS), Jakarta* (pp. 5–10). doi:10.1109/IWBIS.2017.8275095.

Kolias, C., Gritzalis, S., Stavrou, A., & Kambourakis, G. (2016). Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset. *IEEE Communications Surveys & Tutorials, 18*(1), 184–208. doi:10.1109/COMST.2015.2402161.

Li, Y. (2018). Deep reinforcement learning. arXiv:1810.06339v1 [cs.LG].

Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., & Lloret, J. (2019). Shallow neural network with kernel approximation for prediction problems in highly demanding data networks. *Expert Systems with Applications, 124*, 196–208. doi:10.1016/j.eswa.2019.01.063.

Mnih, V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., et al. (2013). Playing ATARI with deep reinforcement learning. arXiv:1312.5602 [cs.LG].

Moshkov, N. (2017). Program for classification of intrusion type on local computer system. Master Thesis. National Research University Higher School of Economics. Moscow. Russia. https://www.hse.ru/en/edu/vkr/206739508.

Nguyen, T.T., & Reddi, V.J. (2019). Deep reinforcement learning for cyber security, arXiv:1906.05799 [cs.CR].

Nisioti, A., Mylonas, A., Yoo, P. D., & Katos, V. (2018). From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys & Tutorials, 20*(4), 3369–3388. doi:10.1109/COMST.2018.2854724.

Nivaashini, M., & Thangaraj, P. (2019). State-of-the-Art machine learning and deep learning: Evolution of intelligent intrusion detection system against wireless network (WiFi) attacks in internet of things (IoT). *International Journal of Innovative Technology and Exploring Engineering (IJITEE), 8*(3).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research, 1*, 2825–2830. http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

Pinto, L., Davidson J., Sukthankar, R., & Gupta A. (2017). Robust adversarial reinforcement learning. arXiv:1703.02702v1 [cs.LG].

Qin, Y., Li, B. Q., & Yan, Z. (2018). Attack detection for wireless enterprise network: A machine learning approach. In *IEEE international conference on signal processing, communications and computing (ICSPCC), Qingdao* (pp. 1–6). doi:10.1109/ICSPCC.2018.8567797.

Rezvy, S., Luo, Y., Petridis, M., Lasebae, A., & Zebin, T. (2019). An efficient deep learning model for intrusion classification and prediction in 5 G and IoT networks. In *2019 53rd annual conference on information sciences and systems (CISS), Baltimore, USA* (pp. 1–6). doi:10.1109/CISS.2019.8693059.

Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). *A survey of network-based intrusion detection data sets* arXiv:1903.02460 [cs.CR].

Servin, A. (2009). *Multi-agent reinforcement learning for intrusion detection* PhD Thesis. UK: University of York.

Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A Deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence, 2*(1), 41–50. doi:10.1109/TETCI.2017.2772792.

Sukhanov, A. V., Kovalev, S. M., & Styskala, V. (2015). (2015). Advanced temporal-difference learning for intrusion detection. *IFAC-PapersOnLine, 48*(4), 43–48. doi:10.1016/j.ifacol.2015.07.005.

Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. *A Bradford book* (2nd ed.) https://mitpress.mit.edu/books/reinforcement-learning-second-edition.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications, Ottawa, ON* (pp. 1–6). 2009. doi:10.1109/CISDA.2009.5356528.

Thanthrige, U. S. K. P. M., Samarabandu, J., & Wang, X. (2016). Machine learning techniques for intrusion detection on public dataset. In *IEEE Canadian conference on electrical and computer engineering (CCECE), Vancouver, BC* (pp. 1–4). doi:10.1109/CCECE.2016.7726677.

Thomas, R., & Pavithran, D. (2018). A survey of intrusion detection models based on NSL-KDD data set. In *2018 Fifth HCT information technology trends (ITT), Dubai, UAE* (pp. 286–291). doi:10.1109/CTIT.2018.8649498.

Tsai, Ch-F., Hsu, Y.-F., Lin, Ch-Y., & Lin, W.-Y. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications., 36*(10), 11994–12000. doi:10.1016/j.eswa.2009.05.029.

Van Hasselt, H., Guez A., Silver D. (2015). Deep reinforcement learning with double Q-learning. arXiv:1509.06461 [cs.LG].

Wang, M., Cui, Y., Wang, X., Xiao, S., & Jiang, J. (2018). Machine learning for networking: Workflow. *Advances and Opportunities. IEEE Network, 32*(2), 92–99. doi:10.1109/MNET.2017.1700200.

Wang, S., Li, B., Yang, M., & Yan, Z. (2019). Intrusion detection for WiFi network: A deep learning approach. international wireless internet conference. WICON 2018: Wireless internet. In *Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering: 264* (pp. 95–104). Cham.: Springer. doi:10.1007/978-3-030-06158-6_10.

Woo, J., Song, J., & Choi, Y. (2019). Performance enhancement of deep neural network using feature selection and preprocessing for intrusion detection. In *2019 International conference on artificial intelligence in information and communication (ICAIIC), Okinawa, Japan* (pp. 415–417). https://doi.org/10.1109/ICAIIC.2019.8668995.

Xu, X. (2006). A sparse kernel-based least-squares temporal difference algorithm for reinforcement learning. *Advances in natural computation. ICNC 2006. lecture notes in computer science*: 4221. Berlin, Heidelberg: Springer. doi:10.1007/11881070_8.

Xu, X., & Luo, Y. (2007). A kernel-based reinforcement learning approach to dynamic behavior modeling of intrusion detection. In *International symposium on neural networks* (pp. 455–464). BerlinHeidelberg: Springer. doi:10.1007/978-3-540-72383-7_54.

Yavanoglu, O., & Aydos, M. (2017). A review on cyber security datasets for machine learning algorithms. In *2017 IEEE international conference on big data (Big data), Boston, MA* (pp. 2186–2193). doi:10.1109/BigData.2017.8258167.

Zarpelo, B. B., Miani, R. S., Kawakani, C. T., & de Alvarenga, S. C. (2017). A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications., 84*, 25–37. doi:10.1016/j.jnca.2017.02.009.